Resolution
Jonathan Buss
©2019, by the author

Resolution is a proof system for propositional logic. Strictly speaking, its use is restricted to formulas in conjunctive normal form (CNF); however, an arbitrary formula is easily converted to CNF before starting a resolution proper.

Recall the definition of CNF: a formula is in CNF if

- it is the conjunction of clauses, where
- each clause is a disjunction of literals, where
- a literal is either a variable or a negated variable.

Due to commutativity and associativity, the order of clauses in a formula or literals in a clause does not matter. Also, no variable need occur twice (or more) in any particular clause. Therefore, one can treat clauses as sets of literals, and formulas as sets of clauses. For example, the CNF formula

\[(p \lor q) \lor p) \land (\neg q \land (r \lor \neg p))\]

is represented as the set containing the clauses

\[\{p, q\}, \{-q\}, \text{ and } \{-p, r\}\ .\]

With this convention, what about the empty set? As a clause, the empty set of literals is taken to have value $F$—it does not contain a true literal. As a formula, the empty set of clauses is taken to have value $T$, although this case rarely occurs.

The empty clause is often denoted by the symbol $\bot$. As we shall see, it has a central role in resolution.

Notions of Proof

Intuitively, a proof is an incontrovertible demonstration that a statement is true.

- No “insight” or “understanding” is required of the reader. Checking the validity of a proof is a purely mechanical process.
- A proof is generally syntactic, rather than semantic. The form, rather than the content, governs. (Semantic reasoning would require some sort of understanding of the meaning.)
- There are inference rules that state which steps of reasoning are permitted. In a sensible proof system, the rules are semantically valid, but their use remains purely syntactic.

---

1The University of Waterloo is granted permission to use this material for teaching and research within the university.

2often called “bottom”, but sometimes “false” or “contradiction”).
Generically, a proof consists of a sequence of formulas, or other statements. (In most cases, a strict sequential order is not required, but rather a partial order instead.)

- The premises, if any, appear first.
- Each formula not among the premises must be a valid inference from preceding formulas.
  That is, there is an inference rule (defined by the proof system) that justifies the formula, based on the previous ones.
- The final formula is the conclusion.

An inference rule describes in what situations a formula may be entered as the next formula of a proof, based on preceding formulas. We notate the statement, “from formulas $\alpha_1, \alpha_2, \ldots, \alpha_n$, infer formula $\beta$” as

$$\frac{\alpha_1 \alpha_2 \ldots \alpha_n}{\beta}$$

Rules are most useful if they specify not specific formulas, but rather a pattern of formulas. For example, a rule specified as $\alpha \beta$ would mean that, for any two formulas already present, we may enter their conjunction next.

Note the syntactic nature of this rule. Given two formulas, one can algorithmically determine that they are actually formulas, and can form their conjunction explicitly.

Informally, outside of the proof system, the example rule might be taken to mean, “if $\alpha$ is true, and $\beta$ is true, then $\alpha \land \beta$ is true”. However, that characterization forms no part of the use of the rule. Notions of true and false don’t enter into using an inference rule.

1. **Definition.** A formula entered into a proof without a justification by a rule is a premise. Normally, premises must appear before other formulas.

   The final formula of a proof is its conclusion.

   If there is a proof with premises $\Sigma$ and conclusion $\varphi$, in which every non-premise follows from preceding formulas via a rule of $S$, then we say that $\varphi$ is provable from $\Sigma$ (in $S$) and write $\Sigma \vdash_S \varphi$ or simply $\Sigma \vdash \varphi$ when $S$ is understood.

OK, but so what? What is the significance of having a proof?

2. **Definition.** A proof system $S$ is sound if, whenever $\Sigma \vdash_S \varphi$, we also have $\Sigma \models \varphi$.

For a sound proof system, every conclusion is actually a logical consequence of its premises.

The converse property to soundness is completeness.

3. **Definition.** A proof system $S$ is complete if, whenever $\Sigma \models \varphi$, there is a proof $\Sigma \vdash_S \varphi$.

Roughly, a system is complete iff every logical consequence has a corresponding proof.
The Resolution proof system

By design, Resolution applies only to CNF formulas. It has essentially only one inference rule:

\[
\frac{\alpha \lor p \quad \neg p \lor \beta}{\alpha \lor \beta}
\]

for any variable \( p \) and formulas \( \alpha \) and \( \beta \). Written as sets, this becomes

\[
\frac{\alpha \cup \{p\} \quad \{\neg p\} \cup \beta}{\alpha \cup \beta}
\]

We consider the following as special cases:

- **Unit resolution** (one of \( \alpha \) or \( \beta \) is the empty set):
  \[
  \frac{\alpha \lor p \quad \neg p}{\alpha}, \text{ i.e., } \frac{\alpha \cup \{p\} \quad \{\neg p\}}{\alpha}
  \]

- **Contradiction** (both \( \alpha \) and \( \beta \) are empty):
  \[
  \frac{p \quad \neg p}{\bot}, \text{ i.e., } \frac{\{p, \neg p\}}{\bot}
  \]

Resolution is a refutation system; a proof is finished when one derives a contradiction \( \bot \). In this case, the original premises are refuted (proven contradictory).

4. Example. To prove: \( \{p, q, \neg p \lor \neg q\} \vdash_{\text{Res}} \bot \).

A proof, as formulas:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( p )</td>
<td>premise</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>( q )</td>
<td>premise</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>( \neg p \lor \neg q )</td>
<td>premise</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>( \neg q )</td>
<td>1, 3</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>( \bot )</td>
<td>2, 4</td>
<td></td>
</tr>
</tbody>
</table>

A proof, as sets:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( {p} )</td>
<td>premise</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>( {q} )</td>
<td>premise</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>( {\neg p, \neg q} )</td>
<td>premise</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>( {\neg q} )</td>
<td>1, 3</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>( \bot )</td>
<td>2, 4</td>
<td></td>
</tr>
</tbody>
</table>

The proof above was shown in linear fashion—simply a sequence of formulas (clauses) and explanations. Often, proofs are displayed more pictorially as a tree or (acyclic) directed graph:

\[
\begin{array}{c}
\{p\} \\
\frac{\{q\} \quad \{\neg p, \neg q\}}{\neg q} \\
\bot
\end{array}
\]

In order to use Resolution successfully, we must account for two features.

- **Resolution only yields contradictions.**

  If we want to prove \( \Sigma \vdash_{\text{Res}} \varphi \), then we must prove \( \Sigma \cup \{\neg \varphi\} \vdash_{\text{Res}} \bot \) instead.
• The resolution rule only applies to disjunctions (\(\lor\)). Thus before applying resolution, first
  – convert each formula to CNF, and
  – separate all formulas at the \(\land\)s.

5. **Example.** To prove: \(\{p, q\} \vdash_{\text{Res}} p \land q\).

Preliminary step 1: move the conclusion to the premises, negating it. This gives premises

\[\{p, q, \neg p \land q\}\]

Preliminary step 2: Convert the premises to CNF:

\[\{p, q, \neg p \lor \neg q\}\]

This is the starting point for the previous example, and the actual proof is the same.

**The Resolution Proof Procedure**

To prove \(\varphi\) from \(\Sigma\), via a Resolution refutation:

1. Convert each formula in \(\Sigma\) to CNF.
2. Convert \(\neg \varphi\) to CNF.
3. Split the CNF formulas at the \(\land\)s, yielding a set of clauses.
4. From the resulting set of clauses, keep applying the resolution inference rule until either:
   - The empty clause \(\bot\) results.  
     In this case, \(\varphi\) is proven from \(\Sigma\).
   - The rule can no longer be applied to give a new formula.  
     In this case, \(\varphi\) cannot be proven from \(\Sigma\).

**Soundness and Completeness of Resolution**

Resolution is sound as a refutation system.

6. **Theorem.** Suppose that \(\Sigma\) is a set of CNF formulas such that \(\Sigma \vdash_{\text{Res}} \bot\). Then \(\Sigma\) is unsatisfiable; i.e., \(\Sigma \models \bot\).

In other words, if we prove something, it is true.

**Proof.** We shall use induction on the length of the resolution proof. We shall begin in the middle, with the inductive step.

7. **Lemma.** Suppose that a set \(\Gamma = \{\beta_1, \ldots, \beta_k\}\) is satisfiable. Let \(\beta_{k+1}\) be a formula obtained from \(\Gamma\) by one use of the resolution inference rule. Then the set \(\Gamma \cup \{\beta_{k+1}\}\) is satisfiable.
Proof of lemma. Let valuation \( v \) satisfy \( \Gamma \); that is, \( \beta_i^v = T \) for each \( i \).

Let \( \beta_{k+1} \) be \( \gamma_1 \lor \gamma_2 \), obtained by resolving \( \beta_i = p \lor \gamma_1 \) and \( \beta_j = \neg p \lor \gamma_2 \).

**Case I:** \( v(p) = F \). Since \( \beta_i^v = T \), we must have \( \gamma_1^v = T \). Thus \( \beta_{k+1}^v = T \).

**Case II:** \( v(p) = T \). Since \( \beta_j^v = T \), we must have \( \gamma_2^v = T \). Thus \( \beta_{k+1}^v = T \).

In either of the two possible cases, we have \( \beta_{k+1}^v = T \), as claimed.

Using induction on \( n \), the previous claim implies

8. **Lemma.** Suppose that the set \( \Gamma = \{ \beta_1, \ldots, \beta_k \} \) is satisfiable. Let \( \alpha \) be a formula obtained from \( \Gamma \) by \( n \) uses of the resolution inference rule. Then the set \( \Gamma \cup \{ \alpha \} \) is satisfiable.

Proof by induction on \( n \).

Therefore, if a set of premises leads to \( \bot \) after any number \( n \) of resolution steps, the set must be unsatisfiable—since any set containing \( \bot \) is unsatisfiable.

Thus Resolution is a sound refutation system, as required.

In some cases, there may be no way to obtain \( \bot \), using any number of resolution steps. What then?

9. **Theorem.** Resolution is a complete refutation system for CNF formulas. That is, if there is no proof of \( \bot \) from a finite set \( \Sigma \) of premises in CNF, then \( \Sigma \) is satisfiable.

Proof. There are only a finite number of clauses possible, over the variables that occur in \( \Sigma \). If one performs resolution in all possible ways, then either \( \bot \) appears, or a fixed-point is reached—that is, every clause that can be formed by the resolution rule already appears in the set of formulas. We refer to such a fixed-point as “closed”.

10. **Lemma.** Suppose that a resolution proof reaches a closed set of clauses, that does not include \( \bot \). Then the entire set of formulas (including the premises) is satisfiable.

Proof of lemma. We use induction on the number of variables present in the formulas.

**Basis:** there are no variables at all—that is, the set of clauses is the empty set. The empty set of clauses is satisfiable, by definition.

**Inductive step:** Suppose that every closed set not containing \( \bot \) of formulas over \( k \) variables is satisfiable.

Consider a closed set of clauses using \( k + 1 \) variables, which does not contain \( \bot \). Select any one variable, say \( p \), and separate the clauses into three sets:

- \( S_p \): the clauses that contain the literal \( p \).
- \( S_{\neg p} \): the clauses that contain the literal \( \neg p \).
- \( R \): the remaining clauses, which do not contain \( p \) at all.
The “remainder” set $R$ has at most $k$ variables. Thus, by hypothesis, $R$ has a satisfying valuation $v$. We consider two cases, according to whether the clauses in $S_p$ are satisfied already by $v$.

**Case I:** Every clause in $S_p$, of the form $p \lor \alpha$, has $\alpha^v = T$.

In this case, the set $S_p$ is already satisfied. Define $v(p) = F$, which additionally makes every clause in $S_{\neg p}$ true.

**Case II:** $S_p$ has some clause $p \lor \alpha$ with $\alpha^v = F$.

In this case, set $v(p) = T$; this satisfies every formula in $S_p$.

Let $\neg p \lor \beta$ be a formula in $S_{\neg p}$. Consider the formula $\alpha \lor \beta$, obtained by resolution from $p \lor \alpha$ and $\neg p \lor \beta$. Since the set is closed, this formula must lie in $R$; thus $\beta^v = T$. Thus also $(\neg p \lor \beta)^v = T$.

In either case, the full set of clauses $S_p \cup S_{\neg p} \cup R$ is satisfiable, as the lemma requires.

By induction, every set that cannot produce $\bot$ is satisfiable, which completes the proof.

**Resolution as an Algorithm**

The resolution method yields an algorithm to determine whether a given formula, or set of formulas, is satisfiable or contradictory.

- Convert to CNF. (A well-specified series of steps.)
- Form resolvents, until either $\bot$ is derived, or no more derivations are possible. (Why must this eventually stop?)
- If $\bot$ is derived, the original formula/set is contradictory. Otherwise, the preceding proof describes how to find a satisfying valuation.

The complexity of this algorithm (both run time and memory space) of course depends on how it is implemented.

- It must handle various sets: each clauses is essentially a set, and there is also the set of clauses itself. The choice of data structure for these sets will affect the run time.

Also, the algorithm must account for where a variable appears, positively and negatively. Again, appropriate data structures help.

- The algorithm must select which of the possible resolutions to do next. Some choices will work better than others. (One strategy is to do resolutions on one variable, until it disappears, and then continue to the remaining ones. In some cases, however, this can dramatically increase the number of steps required.)

Regardless of how “clever” a program is, however, it ultimately produces a Resolution proof—which may need to be quite large.

11. **Theorem (Haken, 1985).**

*There is a number $c > 1$ such that*
For every $n$, there is an unsatisfiable formula on $n$ variables (and about $n^{1.5}$ total literals) whose smallest resolution refutation contains more than $c^n$ steps.

Thus, Resolution is an exponential-time algorithm, in the worst case.

**Resolution in Practice: Satisfiability (SAT) solvers**

Determining the satisfiability of a set of propositional formulas is a fundamental problem in computer science. Examples:

- verification of computer hardware and software,
- scheduling (delivery trucks, airplanes, examinations, ...),
- planning (manufacturing supply chains, building roads, ...),

...many problems of practical importance can be formulated as determining the satisfiability of a set of formulas.

Thus the problem has its own name: SAT. Much effort has been directed toward solving instances of SAT, in practice.

Modern SAT solvers can often solve hard real-world instances with over a million propositional variables and several millions of clauses. However, formulas over a few hundred variables can still defeat the solvers, if the formulas lack regularities.

Currently, almost all programs for large-scale SAT problems make use of resolution. An annual competition is held to identify the current “best” solver. See [http://www.satcompetition.org](http://www.satcompetition.org). Many of the programs listed there are open source; you can play around with them yourself.

**Satisfiability in Theory**

Although the currently best SAT solvers are based on resolution, there is no fundamental reason to believe that resolution is actually the best possible method. Simply put, the basic question is, must any algorithm for SAT use exponential time, in the worst case? Or can we achieve, say $O(n^3)$ time for instances with $n$ distinct variables?

At the center of thinking on this issue lies a simple observation:

Suppose that some formula $\varphi$ is actually satisfiable. Then one can give a short proof of this fact; namely the satisfying valuation. Furthermore, the valuation can be easily checked to be satisfying.

For this course, we shall not delve into precise details. However, the following outlines some of the major ideas.

- A *decision problem* is given by a set of possible *instances* (generally coded as binary strings) and a specification for each instance whether it is to be answered “yes” or “no”.

7
• A decision problem is *solvable in polynomial time* if there is an algorithm that (1) On any instance, yields the correct answer, and (2) runs is time $O(n^c)$ on instances of size $n$, where $c \geq 1$ is a constant for the problem.

(To complete this definition, one needs to define algorithms, and the run-time of an algorithm. We suppress the many details.)

• A decision problem is *verifiable in polynomial time* if each instance with answer “yes” has an associated string, the *witness* for that instance, and also, there is an algorithm that (1a) given a “yes” instance and the associated witness, answers “yes”, (1b) given a “no” instance and any other string whatsoever, answers “no”, and (2) runs in time $O(n^c)$ on instances of size $n$.

(Again, details suppressed.)

• The class of decision problems that are solvable in polynomial time is denoted $P$. The class of decision problems that are verifiable in polynomial time is denoted $NP$.

It is easy to see that a solvable problem is also verifiable (given a witness, simply ignore it and solve the problem). But what about the other direction? Is a verifiable problem always solvable?

12. **Theorem (Cook; Levin).**

*If* $Sat$ *is solvable in polynomial time, then every verifiable problem is. That is,*

$$Sat \in P \iff P = NP.$$  

**Proof sketch.** Suppose one has an algorithm to verify instances for some problem $Q$, which is verifiable. Given an instance $x$ of problem $Q$, one wishes to determine whether or not it has a witness. To to this, create a Boolean formula $\varphi_x$ that says, “There is a witness $w$ s.t. the algorithm calculates the result ‘yes’”. Creating the formula is a matter of writing down exactly what the algorithm does, in full detail.

Now, if one can determine whether the formula $\varphi_x$ is satisfiable, that determines whether $x$ has answer “yes”.

8